HERRAMIENTA GUÍA PARA EL DESARROLLO DESDE EL PUNTO DE VISTA PRÁCTICO DE LOS CURSOS DE INTELIGENCIA ARTIFICIAL Y COMPUTACIÓN BLANDA DE LA UNIVERSIDAD TECNOLÓGICA DE PEREIRA

KAREN TATIANA AGUALIMPIA COPETE
CC. 1088327736
MARIA YESSENIA CASANOVA CASTILLO
CC. 1088303811

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
2018

HERRAMIENTA GUÍA PARA EL DESARROLLO DESDE EL PUNTO DE VISTA PRÁCTICO DE LOS CURSOS DE INTELIGENCIA ARTIFICIAL Y COMPUTACIÓN BLANDA DE LA UNIVERSIDAD TECNOLÓGICA DE PEREIRA

Autor:

KAREN TATIANA AGUALIMPIA COPETE
CC. 1088327736
MARIA YESSENIA CASANOVA CASTILLO
CC. 1088303811

Director:

MSC. JULIO HERNANDO VARGAS MORENO

Proyecto presentado como requisito para optar al título de Ingeniero de Sistemas y Computación

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
2018

Nota de aceptación:

_____
_____
_____
_____
_____

_____
Firma presidente del Jurado

_____
Firma del jurado

_____
Firma del jurado

AGRADECIMIENTOS

En primer lugar, quiero agradecer especialmente al Dios todo poderoso, quien merece la gloria y la honra por siempre y ha sido el motivo de inspiración que mueve mi vida, quien proveyó todo desde un principio para que esto fuese posible.

A mis amados padres, Américo Agualimpia y Angela Copete; mis hermanos, Angela Agualimpia, Mónica Agualimpia y Yeins Agualimpia, y mi novio Dileyson Echeverría. Por tan noble cariño, porque gracias a Dios les permitió estar siempre brindando me su amor, su disposición y apoyo constante e incondicional durante todo este proceso, permitiendo me así alcanzar esta meta.

Quiero agradecer a mis amigos, especialmente a Maria Yessenia Casanoca y a Yeymar Mosquera, quienes contribuyeron de una manera enriquecedora en este arduo camino.

Al profesor Julio Vargas, por compartir sus conocimientos y experiencia, y por la confianza depositada para la realización de este proyecto.

A los profesores que, con su disposición, su experiencia y sus métodos de enseñanza, fueron puntos clave para este crecimiento que sigue en proceso, y que ha permitido hoy, formarme como un profesional integral.

A la Universidad, por ser un lugar que acoge y apoya a todo aquél que lucha por superarse.

De corazón:

¡Muchas Gracias!

Karen Tatiana Agualimpia Copete

AGRADECIMIENTOS

Entre la duda de un camino largo quiero agradecerle primero a Dios por haberme permitido llegar hasta este punto y haberme dado salud para lograr mis objetivos, además de su infinita bondad y amor.

A mis padres por ser el pilar fundamental en todo lo que hoy soy, por su incondicional apoyo perfectamente mantenido a través del tiempo. A mis hermanos que con su infinito amor y dedicación me ayudaron en este recorrido de experiencia y aprendizaje, a mi esposo por el apoyo incondicional en todo momento.

A mi hija por ser esa inspiración ya que con su afecto y cariño es la causa de mi felicidad, de mis esfuerzos, de mis ganas de buscar lo mejor para ti. Te agradezco por ayudarme a encontrar el lado dulce de la vida fuiste mi mayor motivación para concluir con todo este proceso.

A mi compañera de trabajo Karen Tatiana Agualimpia, por su perseverancia y dedicación para realizar este trabajo de grado.

Al profesor Julio Vargas por su gran apoyo y motivación para la culminación de mis estudios profesionales y para la elaboración de este proyecto.

A mis maestros que con perseverancia me enseñaron sus conocimientos e hicieron una profesional de valores, integra y comprometida con su hacer.

A ellos debo agradecer este meta culminado, son ellos quienes deben de gozar de la dicha del triunfo obtenido, de la meta cumplida y lo que continua.

Todo este trabajo ha sido gracias a ellos.

María Yessenia Casanova Castillo

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

1 INTRODUCCIÓN

Algunos de los métodos más interesantes y con importantes aplicaciones en el área de la inteligencia artificial son los métodos llamados de inspiración biológica, dado que se basa en emplear analogías con sistemas naturales o sociales para la resolución de problemas. En la actualidad los "Algoritmos Bioinspirados" son uno de los campos más prometedores de investigación en el diseño de algoritmos, lo que conlleva que el interés en aplicar estos nuevos conocimientos en universidades sea aún mayor. Es por esto que es de vital importancia tener dentro del curso un componente que aborde el tema desde un punto de vista práctico que permita la aplicación de las técnicas de estos algoritmos en la solución de problemas de ingeniería.

Este documento pretende llenar el vacío entre la parte teórica y la parte práctica dentro de los cursos de Inteligencia Artificial (IA) y Computación Blanda (CB) de la Universidad Tecnológica de Pereira. Demostrando cómo una caja de herramientas de aprendizaje de uso general, scikit-learn, puede proporcionar ayuda para que esta unión sea eficiente, ya que es una herramienta de aprendizaje de máquina de referencia y tiene una amplia variedad de algoritmos que se complementan con pocos paquetes, se enfoca en la facilidad de uso, el rendimiento, la documentación y la consistencia API. Tiene dependencias mínimas, es de código abierto y se distribuye bajo la licencia BSD simplificada, lo que fomenta su uso en entornos académicos y comerciales. Pero también porque está implementado en Python, el cual es un lenguaje muy popular para los estudiantes, sin contar con que es fácil de aprender, por ser un lenguaje simple, enfocado en la funcionalidad, que permite que cualquiera, sin necesidad de ser un experto, pueda entender el código; versátil, pues se lleva muy bien con los datos, es uno de los lenguajes más usado en lo que se conoce como ciencia de datos; y por último, es de código abierto y multiplataforma. Por lo tanto, encaja muy bien en un material de desarrollo para las materias antes mencionadas.

El objetivo de este documento no es presentar una biblioteca específica de los algoritmos de scikit-learn, sino más bien una recopilación de los algoritmos que se usan en los cursos de Inteligencia Artificial y Computación Blanda; su usabilidad y sus aplicaciones. En lugar de confiar en una biblioteca de caja negra, preferimos desentrañar ejemplos de código simples y didácticos que permitan a los estudiantes construir sus propias estrategias de análisis.

El principal problema que se plantea es construir una herramienta que, con base a la librería Scikit Learn pueda mostrar la usabilidad de los algoritmos usados dentro del curso de Computación Blanda de la Universidad Tecnológica de Pereira.

2 INSTALANDO SCIKIT-LEARN

Esta herramienta requiere tener instalaciones lo más recientes posibles de:

- scikit-learn
- SciPy
- NumPy
- matplotlib
- pandas
- pillow
- IPython

Si ya tienes una instalación funcional de numpy y scipy, la forma más fácil de instalar scikit-learn es usar pip.

```
pip install -U scikit-learn
```

o conda:

```
conda install scikit-learn
```

3 DATOS EN SCIKIT-LEARN

Los datos en scikit-learn, salvo algunas excepciones, suelen estar almacenados en arreglos de 2 dimensiones, con forma [n_samples, n_features]. Muchos algoritmos aceptan también matrices scipy.sparse con la misma forma.

n_samples: este es el número de ejemplos. Cada ejemplo es un item a procesar (por ejemplo, clasificar). Un ejemplo puede ser un documento, una imagen, un sonido, un vídeo, un objeto astronómico, una fila de una base de datos o de un fichero CSV, o cualquier cosa que se pueda describir usando un conjunto prefijado de trazas cuantitativas.
n_features: éste es el número de características descriptoras que se utilizan para describir cada ítem de forma cuantitativa. Las características son, generalmente, valores reales, aunque pueden ser categóricas o valores discretos.

El número de características debe ser fijado de antemano. Sin embargo, puede ser extremadamente alto (por ejemplo, millones de características), siendo cero en la mayoría de los casos. En este tipo de datos, es buena idea usar matrices scipy.sparse que manejan mucho mejor la memoria.

Se utilizarán cuatro conjuntos de datos: iris y digit para la clasificación, diabetes y boston para la regresión.

Datos Iris

Características del conjunto de datos iris:

1. Longitud del sépalo en cm.
2. Ancho del sépalo en cm
3. Longitud del pétalo en cm
4. Ancho del pétalo en cm

Clases objetivo que predecir:

1. clase 0: Iris Setosa
2. clase 1: Iris Versicolor
3. clase 3: Iris Virginica

Clases	3
Muestras por clase	50
Muestras totales	150
Dimensionalidad	4
Características	real, positive

Tabla 1. *Detalles del conjunto de datos Iris*

Datos Digit

Dígitos es un conjunto de datos de dígitos escritos a mano. Cada característica es la intensidad de un píxel de una imagen de 8 x 8.

Clases	10
Muestras por clases	~180
Muestras totales	1797
Dimensionalidad	64
Características	integer 0-16

Tabla 2. *Detalles del conjunto de datos Digit*

Datos Boston

Boston es un conjunto de datos de precios de viviendas de Boston.

Muestras totales	506
Dimensionalidad	13
Características	real, positive
Objetivos	real 5 – 50

Tabla 3. *Detalles del conjunto de datos Boston*

Datos Diabetes

Diabetes es un conjunto de datos sobre pacientes con diabetes.

Muestras totales	442
Dimensionalidad	10
Características	real, -2 < x < 2
Objetivos	integer 25 - 0346

Tabla 4. *Detalles del conjunto de datos Diabetes*

3.1 Cargar datos

Cargar datos.

```
from sklearn.datasets import load_iris
iris = load_iris()
```

Para ver el contenido disponible de los datos

```
Print(iris.keys())
```

Como resultado:

```
['target_names', 'data', 'target', 'DESCR', 'feature_names']
```

Los nombres de las clases se almacenan en el último atributo, target_names.

```
print(iris.target_names)
```

Las características de cada muestra de flor se almacenan en el atributo *data* del conjunto de datos:

```
n_samples, n_features = iris.data.shape
print('Number of samples:', n_samples)
print('Number of features:', n_features)
print(iris.data[0])
```

```
('Number of samples:', 150)
('Number of features:', 4)
```

la longitud del sépalo, el ancho del sépalo, la longitud del pétalo y el ancho del pétalo de la primera muestra de la primera flor.

```
[5.1 3.5 1.4 0.2]
```

La información sobre las clases de cada muestra está almacena en el atributo *target* del conjunto de datos:

```
print(iris.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

4. GRADIENTE DESCENDENTE ESTOCASTICO

El **Descenso de Gradientes Estocástico** (SGD) es un enfoque simple pero muy eficiente para el aprendizaje discriminatorio de clasificadores lineales bajo funciones de pérdida convexa tales como Máquinas Vectoriales de Soporte (lineal) y Regresión Logística.

Clase SGDClassifier:

***class* sklearn.linear_modgel.SGDClassifier**(*loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, warm_start=False, average=False, n_iter=None*)

Parámetros	Descripción
X	Conjunto de datos.
y	Etiquetas para **X.**

Parámetros	Descripción
loss	str, valor predeterminado = ‘hinge’ La función de pérdida que se utilizará, indica la función de error que se trata de minimizar. Las opciones posibles: “hinge”, “log”, “modified_huber”, 'squared_hinge', 'perceptron', o una pérdida de regresión: 'squared_loss', 'huber', 'epsilon_insensitive' o 'squared_epsilon_insensitive'.
penalty	str, ‘none’, ‘l2’, ‘l1’, o ‘elasticnet’ El ajuste predeterminado es penalty=”l2″. La penalización L1 conduce a soluciones escasas, llevando la mayoría de los coeficientes a cero. La Red Elástica resuelve algunas deficiencias de la penalización L1 en presencia de atributos altamente correlacionados. El parámetro l1_ratio controla la combinación convexa de penalización L1 y L2.
alpha	float Constante que multiplica el término de regularización. El valor predeterminado es 0.0001 También se usa para calcular la tasa de aprendizaje cuando se establece en “optimal”.
l1_ratio	float El parámetro de mezcla Elastic Net, con 0 <= l1_ratio <= 1. l1_ratio = 0 corresponde a la penalización de L2, l1_ratio = 1 a L1. El valor predeterminado es 0.15.
fit_intercept	bool, valor predeterminado = True. Si la intercepción debe ser estimada o no. Si es falso, se supone que los datos ya están centrados.
intercept_init	Array de la forma (n_classes,) El intercepto inicial para la optimización de warm-start.
max_iter	int, opcional El número máximo de pases sobre los datos de entrenamiento (aka épocas). Solo afecta el comportamiento en el método *fit*, y no el *partial_fit*. El valor predeterminado es 5. Predeterminado a 1000 desde 0.21, o si *tol* no es None.
tol	float o None, opcional El criterio de parada. Si no es None, las iteraciones se detendrán cuando (loss> previous_loss - tol). El valor predeterminado es None. El valor predeterminado es 1e-3 desde 0.21.
shuffle	bool, opcional, el valor predeterminado es True. Si los datos de entrenamiento deben ser mezclados después de cada época.
verbose	integer, opcional Nivel de verbosidad.

Parámetros	Descripción
epsilon	float Épsilon en las funciones de pérdida insensibles a épsilon; solo si la pérdida es 'huber', 'epsilon_insensitive' o 'squared_epsilon_insensitive'. Para 'huber', determina el umbral en el que se vuelve menos importante obtener la predicción exactamente correcta. Para insensible a épsilon, cualquier diferencia entre la predicción actual y la correcta se ignora si es menor que este umbral.
n_jobs	int, opcional La cantidad de CPU que se utilizará para realizar el cálculo de OVA (Uno contra todos, para problemas de clase múltiple). -1 significa 'todas las CPU'. El valor predeterminado es 1.
random_state	int, instancia de RandomState o None, opcional, valor predeterminado: None La semilla del generador de números pseudoaleatorios para usar al mezclar los datos. Si int, random_state es la semilla utilizada por el generador de números aleatorios; Si la instancia de RandomState, random_state es el generador de números aleatorios; Si no es así, el generador de números aleatorios es la instancia de RandomState utilizada por np.random .
learning_rate	string, opcional El programa de tasa de aprendizaje: • 'constant': eta = eta0 • 'optimal': eta = 1.0 / (alpha * (t + t0)) [valor predeterminado] • 'invscaling': eta = eta0 / pow(t, power_t) 'donde t0 es elegido por una heurística propuesta por Leon Bottou.
eta0	double La tasa de aprendizaje inicial para los programas 'constant' o 'invscaling'. El valor predeterminado es 0.0 ya que eta0 no es utilizado por la programación predeterminada 'optimal'.
power_t	double El exponente para la velocidad de aprendizaje de escala inversa [valor predeterminado 0.5].
class_weight	dic, {class_label: weight} o "balanced" o None, opcional Preestablecido para el parámetro de ajuste class_weight. Pesos asociados con las clases. Si no se da, se supone que todas las clases tienen peso uno. El modo "balanced" usa los valores de y para ajustar automáticamente los pesos inversamente proporcionales a las frecuencias de clase en los datos de entrada como n_samples / (n_classes * np.bincount(y))
classes	Array de la forma (n_classes,) Clases en todas las llamadas a partial_fit. Puede obtenerse a través de np.unique (y_all) , donde y_all es el vector de destino de todo el conjunto de datos. Este argumento es necesario para la primera llamada a partial_fit y se puede omitir en las llamadas siguientes. Tenga en cuenta que y no necesita contener todas las etiquetas en las clases .

Parámetros	Descripción
samplee_weight	Array de la forma (n_samples,), opcional Pesos aplicados a muestras individuales. Si no se proporciona, se suponen los pesos uniformes.
warm_start	boolean, valor predeterminado True Cuando se establece en True, reutilice la solución de la llamada anterior para que se ajuste como inicialización, de lo contrario, simplemente borre la solución anterior.
average	bool o int, opcional Cuando se establece en True, calcula los pesos SGD promediados y almacena el resultado en el atributo coef_. Si se establece en un entero mayor que 1, el promedio comenzará una vez que el número total de muestras vistas alcance el promedio. Entonces average=10 comenzará el promedio después de ver 10 muestras.
n_iter	int, opcional El número de iteraciones sobre los datos de entrenamiento (aka epochs). El valor predeterminado es None.
coef_init	Array de la forma (n_classes, n_features) Los coeficientes iniciales para la optimización de warm_start.

Tabla 5. *Parámetros de la clase SGDClassifier - SGDRegressor*

Atributos	Descripción
coef_	Array de la forma (1, n_features) if n_classes == 2 else (n_classes, n_features) Peso asignado a las características.
intercepts_	Array de la forma (1,) if n_classes == 2 else (n_classes,) Constantes en la función de decisión.
n_iter_	int El número real de iteraciones para alcanzar el criterio de detención. Para ajustes multiclase, es el máximo sobre cada ajuste binario
loss_function_	LossFunction

Tabla 6. *Atributos de la clase SGDClassifier*

Métodos	Descripción
decision_function(X)	Predice los puntajes de confianza para las muestras
densify()	Convertir matriz de coeficientes a formato de matriz densa
fit(X, y[, coef_init, intercept_init, …])	Ajustar modelo lineal con pendiente de gradiente estocástico
get_params([deep])	Obtiene los parámetros para este estimador. **deep:** boolean, opcional (deep=True)
partial_fit(X, y[, classes, sample_weight])	Ajustar modelo lineal con pendiente de gradiente estocástico
predict(X)	Predecir etiquetas de clase para muestras en X.
score(X, y[, sample_weight])	Devuelve la precisión media en los datos y etiquetas de prueba dado.
set_params(*args, **kwargs)	Establece los parámetros del estimador.
sparsify()	Convierte la matriz de coeficientes a formato disperso

Tabla 7. *Métodos de la clase SGDClassifier*

4.1 Clasificación del gradiente descendente estocástico

4.1.1 Ejemplo 1

Cargar librerías

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
```

```
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
y = np.array([1, 1, 2, 2])
```

Se ajusta el modelo y se le indica que use una función de pérdida "hinge" con una penalidad l2. Esta función de pérdida es perezosa, sólo actualiza los parámetros del modelo si un ejemplo viola la restricción de margen, lo que hace que el entrenamiento sea muy eficiente y puede resultar en modelos de sparser, incluso cuando se usa la penalización L2.

```
clf = SGDClassifier(loss="hinge", penalty="l2")
```

Se entrena el modelo.

```
clf.fit(X, y)
```

Se predicen nuevos valores.

```
print(clf.predict([[3.,2.]]))
```

```
array([0])
```

Se mantienen los parámetros del modelo.

```
clf.coef_
```

```
array([[-49.30966469, -49.30966469],
       [ -9.86193294,  -9.86193294],
       [  9.86193294,   9.86193294]])
```

Se mantiene la intercepción.

```
clf.intercept_
```

El uso de loss=”log” habilita el método predict_proba(), que proporciona un vector de estimaciones de probabilidad $P(y|X)$ por muestras X.

```
clf =SGDClassifier(loss="log").fit(X,y)
clf.predict_proba([[1.,1.]]
```

```
array([[-236.60806473,  -49.44782823,    9.53785038]])
```

Se puede ver que el algoritmo aprendió demasiado bien, con una predicción del 100%.

```
print(clf.score(X, y))
```

```
1.0
```

4.1.2 Ejemplo 2

Trazar la superficie de decisión del SGD de clase múltiple en el conjunto de datos del iris. Los hiperplanos correspondientes a los tres clasificadores de uno contra todos (OVA) están representados por las líneas punteadas.

Cargar librerías.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import SGDClassifier
```

Se cargan los datos.

```
iris = datasets.load_iris()
```

Solo tomamos las dos primeras características. Podríamos evitar este feo corte utilizando un conjunto de datos de dos dimensiones.

```
X = iris.data[:, :2]
y = iris.target
colors = "bry"
```

Esta función *random.shuffle()* solo mezcla la matriz a lo largo del primer eje de una matriz multidimensional. El orden de las sub-matrices cambia, pero su contenido permanece igual.

```
# shuffle
idx = np.arange(X.shape[0])
np.random.seed(13)
np.random.shuffle(idx)
X = X[idx]
y = y[idx]
```

Se traza la línea, los puntos y los vectores más cercanos al plano.

```
# standardize
mean = X.mean(axis=0)
std = X.std(axis=0)
X = (X - mean) / std
```

Tamaño de paso en la malla.

```
h = .02
```

Se entrena el modelo.

```
clf = SGDClassifier(alpha=0.001, max_iter=100)
```

Se ajusta el modelo.

```
clf.fit(X, y)
```

Crea una malla para trazar.

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
```

Trace el límite de decisión. Para eso, asignaremos un color a cada.
Punto en la malla [x_min, x_max]x[y_min, y_max]

```
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

Pon el resultado en una trama de color

```
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.axis('tight')
```

Trazar también los puntos de entrenamiento.

```
for i, color in zip(clf.classes_, colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=color,
                label=iris.target_names[i],
                cmap=plt.cm.Paired, edgecolor='black', s=20)
plt.title("Decision surface of multi-class SGD")
plt.axis('tight')
```

Grafíca los tres clasificadores uno contra todos.

```
xmin, xmax = plt.xlim()
ymin, ymax = plt.ylim()
coef = clf.coef_
intercept = clf.intercept_

def plot_hyperplane(c, color):
    def line(x0):
        return (-(x0 * coef[c, 0]) - intercept[c]) / coef[c, 1]

    plt.plot([xmin, xmax], [line(xmin), line(xmax)],
             ls="--", color=color)

for i, color in zip(clf.classes_, colors):
    plot_hyperplane(i, color)
plt.legend()
plt.show()
```

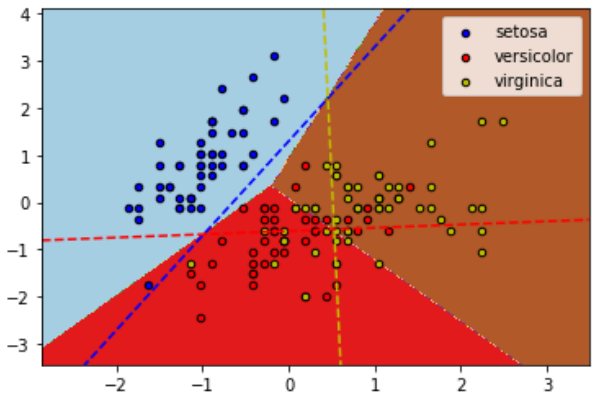

Figura 1. *Superficie de decisión del SGD multiclase*

5. PERCEPTRÓN SIMPLE

El perceptrón es un algoritmo apto para el aprendizaje a gran escala**.**

El perceptrón es otro algoritmo simple adecuado para el aprendizaje a gran escala. Valor predeterminado:

- No requiere una tasa de aprendizaje.
- No está regularizado (penalizado).
- Actualiza su modelo solo en errores.

Clase Perceptron:

class sklearn.linear_model.**Perceptron**(*penalty=None, alpha=0.0001, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, eta0=1.0, n_jobs=1, random_state=0, class_weight=None, warm_start=False, n_iter=None*)

Parámetros	Descripción
X	Conjunto de datos
y	Etiquetas para **X**
penalty	None, 'l2' or 'l1' o 'elasticnet, valor predeterminado: None La pena (también conocido como término de regularización) que se utilizará.
alpha	float, el valor predeterminado es 0.0001 Constante que multiplica el término de regularización en caso de que se use una regularización.
fit_intercept	bool, el valor predeterminado es True Si la intercepción debe ser estimada o no. Si es falso, se supone que los datos ya están centrados.
max-iter	int, opcional El número máximo de pases sobre los datos de entrenamiento solo afecta el comportamiento en el método ***fit,*** y no el ***partial_fit***. El valor predeterminado es 5. Predeterminado a 1000 desde 0.21, o si el parámetro ***tol*** no es None.
tol	float o None, opcional El criterio de parada. Si no es None, las iteraciones se detendrán cuando (loss>previous_loss-tol). El valor predeterminado es None. El valor predeterminado es 1e-3 desde 0.21
shuffle	bool, opcional, valor predeterminado = True Si los datos de entrenamiento deben ser mezclados después de cada época.
verbose	integer, opcional Nivel de verbosidad.
eta0	double Si los datos de entrenamiento deben ser mezclados después de cada época.

Parámetros	Descripción
n_jobs	integer, opcional La cantidad de CPU que se utilizara para realizar el cálculo de OVA (uno contra todos, para problemas de clases múltiples). -1 significa 'todas la CPU'. El valor predeterminado es 1.
random_state	int, instancia de RamdomState o None, opcional, valor predeterminado = None La semilla del generador de números pseudoaleatorios para usar al mezclar los datos. Si int, random_state es la semilla utilizada por el generador de números aleatorios; si la instancia de RandomState, random_state es el generador de números aleatorios; si no es asi, el generador de números aleatorios es la instancia de RandomState utilizada por np.random.
class-weight	dict, {class_label: weight} o "balance" o None, opcional Preestablecido para el parámetro class_weight fit. Pesos asociados con las clases. Si no se da, se supone que todas las clases tienen peso uno.
warm-start	bool, opcional Cuando se establece en True, reutilice la solución de la llamada anterior para que se ejecute como inicialización, de lo contrario, simplemente borre la solución anterior.
n_iter	int, opcional El número de pases sobre los datos de entrenamiento. El valor predeterminado es None. Obsoleto se eliminará en 0.21.

***Tabla 8**. Parámetros de la clase Perceptron*

Atributos	Descripción
coef_	array, de la forma [1, n_features] si n_classes == 2 else [n_classes, n_features] Pesos asignados a las características
intercept_	array de la forma = [1] si n_classes == 2 else [n-classes] Constante en la función de decisión
n_inter_	int El número real de iteraciones para alcanzar el criterio de detención. Para ajustar multiclase, es el máximo sobre cada ajuste binario.

***Tabla 9.** Atributos de la clase Perceptron*

Métodos	Descripción
decision_function(X)	Predice los puntajes de confianza para las muestras.
densify()	Convertir matriz de los coeficientes a formato de matriz densa.
fit(X, y[coef_init, intercept_init,..])	Ajustar modelo lineal con pendiente de gradiente estocástico.
get_params([deep])	Obtenga los parámetros para este estimador.
partial_fit(X,y[sample_weight])	Ajustar modelos lineales con pendiente de gradiente estocástico.
predict(X**)**	Predecir etiquetas de clase para muestras en X.
score(X, y[sample_weight])	Devuelve la precisión media en los datos y etiqueta de prueba datos.
set_params(*sample_weight)	Establece los parámetros del estimador.
sparsify()	Convierte la matriz de coeficiente a formato disperso.

Tabla 10. *Métodos de la clase Perceptron*

5.1 Ejemplo 1

Cargar librerías

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
```

Cargar datos.

```
iris = datasets.load_iris()
```

```
X = iris.data
y = iris.target
```

Ver las primeras cinco observaciones de y.

```
y[:5]
```

```
array([0, 0, 0, 0, 0])
```

Ver las primeras cinco observaciones de X.

```
X[:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
```

```
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

Separar los datos de entrenamiento y prueba.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.3)
```

Se crea un objeto perceptrón con los parámetros: 40 iteraciones sobre los datos, y una tasa de aprendizaje de 0.1.

```
ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)
```

Se entrena el perceptrón.

```
ppn.fit(X_train_std, y_train)
```

Se aplica el perceptrón entrenado en los datos de X para hacer predicciones para los datos de prueba y.

```
y_pred = ppn.predict(X_test_std)
```

Ver los datos predichos de prueba y.

```
y_pred
```

```
array([2, 2, 0, 2, 2, 2, 0, 0, 2, 0, 2, 0, 2, 2, 1, 2, 0, 1, 1, 1,
2, 0, 2, 0, 2, 2, 2, 0, 2, 1, 2, 0, 2, 0, 0, 1, 0, 0, 2, 2, 1, 2,
2, 2, 1])
```

Ver los datos reales de prueba y.

```
y_test
```

```
array([1, 2, 0, 2, 2, 2, 0, 0, 2, 0, 2, 0, 2, 2, 1, 2, 0, 1, 2, 1,
1, 0, 2, 0, 2, 2, 2, 0, 1, 1, 2, 0, 1, 0, 0, 1, 0, 0, 2, 2, 1, 2,
2, 2, 1])
```

Ver la precisión del modelo, que es: 1 - (observaciones predicen observaciones incorrectas / totales).

```
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.89
```

5.2 Ejemplo 2

Un ejemplo que muestra cómo diferentes solucionadores en línea realizan en el conjunto de datos de dígitos escritos a mano.

Cargar librerías.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier, Perceptron
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.linear_model import LogisticRegression
```

Cargar datos.

```
heldout = [0.95, 0.90, 0.75, 0.50, 0.01]
rounds = 20
digits = datasets.load_digits()
X, y = digits.data, digits.target
```

Modelos de clasificación a entrenar.

```
classifiers = [
    ("SGD", SGDClassifier()),
    ("ASGD", SGDClassifier(average=True)),
    ("Perceptron", Perceptron()),
    ("Passive-Aggressive I",
     PassiveAggressiveClassifier(loss='hinge', C=1.0)),
    ("Passive-Aggressive II",
     PassiveAggressiveClassifier(loss='squared_hinge', C=1.0)),
    ("SAG", LogisticRegression(solver='sag', tol=1e-1,
                               C=1.e4 / X.shape[0]))
]
```

Se entrenan los modelos.

```
xx = 1. - np.array(heldout)

for name, clf in classifiers:
    print("training %s" % name)
    rng = np.random.RandomState(42)
    yy = []
    for i in heldout:
        yy_ = []
        for r in range(rounds):
            X_train, X_test, y_train, y_test = \
               train_test_split(X, y, test_size=i, random_state=rng)
            clf.fit(X_train, y_train)
```

```
            y_pred = clf.predict(X_test)
            yy_.append(1 - np.mean(y_pred == y_test))
        yy.append(np.mean(yy_))
    plt.plot(xx, yy, label=name)
```

Se grafican los resultados.

```
plt.xlabel("Proportion train")
plt.ylabel("Test Error Rate")
plt.show()
```

```
training SGD
training ASGD
training Perceptron
training Passive-Aggressive I
training Passive-Aggressive II
training SAG
```

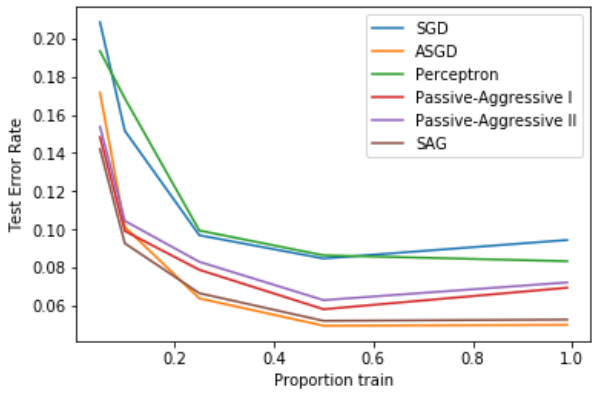

***Figura 2**. Comparando varios solucionadores en línea*

6 PERCEPTRÓN MULTICAPA

Clase MLPClassifier

class sklearn.neural_network.**MLPClassifier**(*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08*)

Parámetros	Descripción
X	Conjunto de datos
y	Etiquetas para **X**
hidden_layer_sizes	tuple, length = n_layers – 2, default (100) El i-ésimo elemento representa el número de neuronas en la i-ésima capa oculta
activation	{'identity', 'logistic', 'tanh', 'relu'}, default 'relu' Función de activación para la capa oculta ▪ 'identity', activación no operativa, útil para implementar el cuello de botella lineal, devuelve f(x) = x ▪ 'logistic', la función logística sigmoidea, devuelve f (x) = 1/ (1 + exp (-x)) ▪ 'tanh', la función de saturación hiperbólica devuelve f (x) = tanh (x) 'relu', la función rectificada de la unidad lineal devuelve f (x) = max (0, x)
solver	{'lbfgs', 'sgd', 'adam'}, valor predeterminado 'adam' El solucionador para la optimización del peso ▪ 'lbfgs', es un optimizador en la familia de métodos quasi-Newton ▪ 'sgd', se refiere al descenso de gradiente estocástico ▪ 'adam', se refiere a un optimizador estocástico basado en gradiente propuesto por Kingma, Diederik y Jimmy Ba.
alpha	float. Opcional, valor predeterminado 0.001 Parámetro L2 de penalización (término de regulación).
batch_size	int, opcional, predeterminado 'auto' Tamaño de mini batches para optimizadores estocásticos. Si el solucionador es 'lbfgs' el clasificador no usará mini batch. Cuando se establecen en "aut"', batch_size = min (200, n_samples)
learning_rate	{'constant', 'invscaling', 'adaptive'}, valor predeterminado 'constant' Programa de tasa de aprendizaje para actualizaciones de peso ▪ 'constant', tasa de aprendizaje constante dada por 'learning_rate_init' ▪ 'invscaling', disminuye gradualmente la tasa de learning_rate_ en casa paso de tiempo 't' usando un exponente de escala inversa de 'power_t' effective_learning_rate = learning_rate_init / pow (t, power_t) ▪ 'adaptive' mantiene constante la tasa de aprendizaje a 'learning_rate_init' siempre que la perdida de entrenamiento siga disminuyendo. Solo se usa cuando solver = 'sgd'
learning_rate_int	double, opcional, valor predeterminado = 0.001 La tasa de aprendizaje inicial utilizada controla el tamaño del paso al actualizar los pesos. Solo se usa cuando solucionador = 'sgd' o 'adam'.
power_t	double, opcional, valor predeterminado 0.5 El exponente de velocidad de aprendizaje de escala inversa. Se usa para actualizar la tasa de aprendizaje efectiva cuando el valor de aprendizaje se establece en 'invscaling'. Solo se usa cuando solucionador = 'sgd'

Parámetros	Descripción
max_iter	int, opcional, valor predeterminado 200 Número máximo de iteraciones El solucionador itera hasta la convergencia (determinada por 'tol') o este número de iteraciones. Para los solucionadores estocásticos ('sgd', 'adam'), tenga en cuenta que esto determina el número de épocas (cuántas veces se usará cada punto de datos), no el número de pasos de gradiente.
shuffle	bool, opcional, default True Si mezclar muestras en cada iteración. Solo se usa cuando solucionador = 'sgd' o 'adam'.
random_state	int, instancia de RandomState o None, opcional, por defecto None Si int, random_state es la semilla utilizada por el generador de números aleatorios; Si la instancia de RandomState, random_state es el generador de números aleatorios; Si no es así, el generador de números aleatorios es la instancia de RandomState utilizada por np.random
tol	float, opcional, default 1e-4 Tolerancia para la optimización. Cuando la pérdida o el puntaje no mejoran al menos tol durante dos iteraciones consecutivas, a menos que learning_rate se configure como 'adaptativo', se considera que se ha alcanzado la convergencia y se detiene el entrenamiento.
verbose	bool, opcional, valor predeterminado False Si se deben imprimir los mensajes de progreso a stdout.
warm-start	bool, opcional, default False Cuando se establece en True, reutilice la solución de la llamada anterior para que se ajuste como inicialización, de lo contrario, simplemente borre la solución anterior.
momentum	float, default 0.9 Momentum para la actualización de descenso de gradiente. Debe estar entre 0 y 1. Solo se usa cuando solucionador = 'sgd'.
nesterovs_momentum	boolean, predeterminado True Si usar el impulso de Nesterov. Solo se usa cuando resolve = 'sgd' y momentum> 0.
early_stopping	bool, valor predeterminado False Ya sea para usar la detención anticipada para finalizar la capacitación cuando la puntuación de validación no está mejorando. Si se establece en verdadero, dejará de lado automáticamente el 10% de los datos de entrenamiento como validación y finalizará el entrenamiento cuando el puntaje de validación no mejore al menos tol durante dos épocas consecutivas. Solo es efectivo cuando solucionador = 'sgd' o 'adam'
validation_fraction	float, opcional, predeterminado 0.1 La proporción de datos de entrenamiento para separar como conjunto de validación para la detención temprana. Debe estar entre 0 y 1. Solo se usa si early_stopping es True
beta_1	float, opcional, predeterminado 0.9

Parámetros	Descripción
	La tasa de desintegración exponencial para las estimaciones del vector de primer momento en adam, debe estar en [0, 1). Solo se usa cuando solucionador = 'adam'
beta_2	float, opcional, default 1e-8 Valor para la estabilidad numérica en adam. Solo se usa cuando solucionador = 'adam'
epsilon	float, opcional, default 1e-8 Valor para la estabilidad numérica en adam. Solo se usa cuando solucionador = 'adam'

Tabla 11. *Parámetros de la clase MLPClassifier*

Atributos	Descripción
classes_	array o lista de array de la forma (n_classes,) Etiquetas de clase para cada salida.
loss_	float La pérdida actual calculada con la función de pérdida.
coefs_	list, length n_layers -1 El i-ésimo elemento en la lista representa la matriz de ponderación correspondiente a la capa i.
intercepts_	list, length n_layers – 1 El i-ésimo elemento en la lista representa el vector de polarización correspondiente a la capa i + 1.
n_inter_	int La cantidad de iteraciones que el solucionador ejecutó.
n_layers_	int Número de capas.
n_outputs_	int Cantidad de salidas.
out_activation_	string Nombre de la función de activación de salida.

Tabla 12. *Atributos de la clase MLPClassifier*

Métodos	Descripción
fit(X, y)	Ajuste el modelo a la matriz de datos X y al objetivo (s) y.
get_params([deep])	Obtiene los parámetros para este estimador. **deep:** boolean, opcional (deep=True)
predict(X)	Predecir usando el clasificador de perceptrón multicapa.
predict_log_proba(X)	Devuelve el registro de las estimaciones de probabilidad.
predict_proba(X)	Estimaciones de probabilidad de retorno para los datos de prueba X.
score(X, y[, sample_weight])])	Devuelve la precisión media en los datos y etiquetas de prueba dados.
set_params(** params)	Establece los parámetros del estimador.

Tabla 13. *Métodos de la clase MLPClassifier*

6.1 Ejemplo 1

Cargar librerías.

```
from sklearn.neural_network import MLPClassifier
```

Crear datos.

```
X = [[0., 0.], [1., 1.]]
y = [0, 1]
```

Se entrena el modelo.

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)
```

Se ajusta el modelo.

```
clf.fit(X, y)
```

Se predicen nuevos modelos.

```
clf.predict([[2., 2.], [-1., -2.]])
```

```
array([1, 0])
```

Contiene las matrices de peso que construyen los parámetros del modelo.

```
[coef.shape for coef in clf.coefs_]
```

```
[(2, 5), (5, 2), (2, 1)]
```

Actualmente, MLPClassifiersolo admite la función de pérdida Cross-Entropy, que permite estimaciones de probabilidad al ejecutar el método predict_proba.

MLP entrena usando Backpropagation. Más precisamente, se entrena usando alguna forma de descenso de gradiente y los gradientes se calculan utilizando Backpropagation. Para la clasificación, minimiza la función de pérdida Cross-Entropy, dando un vector de estimaciones de probabilidad *P(y | x)* por muestra X.

```
clf.predict_proba([[2., 2.], [1., 2.]])
```

```
array([[1.96718015e-04, 9.99803282e-01],
       [1.96718015e-04, 9.99803282e-01]])
```

6.2 Ejemplo 2

Ejemplo de reconocimiento de dígitos escritos a mano.

Cargar librerías.

```
import scipy.io as sio
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import StratifiedShuffleSplit, GridSea
rchCV
```

Cargar datos.

```
digits = sio.loadmat('handwritten.mat.mat')
```

Datos de imagen de dígitos (5000 imágenes con 400 características / píxeles.

```
X = digits['X']
```

Clases de dígitos (1-10) donde se asigna el dígito 0 clase 10.

```
y = digits['y'].ravel()
```

Cree una red neuronal de prealimentacón con una sola capa oculta con 25 unidades logísticas. El tamaño de la capa de entrada se deriva del número de características en X. El tamaño de la capa de salida se deriva del número de clases en y.

```
clf = MLPClassifier((25,), activation='logistic', solver='lbfgs')
```

Se ajusta el modelo

```
clf.fit(X, y)
```

Exactitud de clasificación en el conjunto de entrenamiento.

```
clf.score(X, y)
```

Para la validación cruzada, utilice dos pliegues aleatorios estratificados en los que el tamaño del conjunto de prueba sea igual a 0.1 * tamaño del conjunto de datos. Estratificado significa que los pliegues se realizan conservando el porcentaje de muestras para cada clase.

```
cv = StratifiedShuffleSplit(2, test_size=0.1, random_state=0)
```

Ejecute una búsqueda en cuadrícula para encontrar los mejores valores de valor para el parámetro de regularización alfa usando el validador cruzado (cv) definido anteriormente.

```
gs = GridSearchCV(clf, param_grid={'alpha':[1e-2, 1e-1, 1e0, 1e1]},
cv=cv)
gs.fit(X, y)
```

El mejor clasificador es la red neuronal con alfa = 1.0.

```
clf_best = gs.best_estimator_
clf_best
```

Exactitud de clasificación en el conjunto de entrenamiento.

```
clf_best.score(X, y)
```

```
0.99739999999999995
```

Obtener la matriz de peso entre la entrada y la capa oculta (tamaño de capa de entrada = 400, tamaño de capa oculta = 25).

```
Theta1 = clf_best.coefs_[0].T
Theta1.shape
```

```
(25, 400)
```

Visualice los pesos de entrada en unidades ocultas (400 por unidad oculta).

```
n_rows = 5
n_cols = 5

plt.subplots_adjust(top=.9, hspace=.4)
plt.figure(figsize=(1.8 * n_cols, 2.4 * n_rows))
```

```
for i, row in enumerate(Theta1):
    plt.subplot(n_rows, n_cols, i + 1)
    plt.imshow(row.reshape((20,20), order='F'), cmap=plt.cm.gray)
    plt.title(f'hidden unit {i + 1}')
    plt.xticks(())
    plt.yticks(())
```

***Figura 3**.Reconocimiento de dígitos escritos a mano*

7 REGRESIÓN LINEAL

En su forma más simple, ajusta un modelo lineal al conjunto de datos ajustando un conjunto de parámetros para hacer que la suma de los residuos cuadrados del modelo sea lo más pequeña posible.

Clase LinearRegression

Class sklearn.linear_model.**LinearRegression***(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)*

Parámetros	Descripción
fit_intercept	boolean, opcional, el valor predeterminado es True Si se establece en False, no se usará ninguna intercepción en los cálculos (Se espera que los datos ya estén centrados.
normalize	boolean, opcional, el valor determinado es False Se ignora cuando fit_intercept se establece en False. Si es verdadero, los regresores X se normalizarán antes de la regresión restando la media y dividiendo por la norma l2. Si desea estandarizar, use sklearn.preprocessing.StandardScaler antes de llamar a fit en un estimador con normalize=False
copy_X	boolean, opcional, predeterminado True Si es verdadero, X se copiará; de lo contrario, puede sobrescribirse.
n_jobs	int, opcional, el valor predeterminado es 1 El número de trabajos para usar para el cálculo. Si -1 se usan todas las CPU. Esto solo proporcionará aceleración para n_targets> 1 y suficientes problemas grandes.

Tabla 14. *Parámetros de la clase LinearRegression*

Atributos	Descripción
coef_	Array de la forma (n_features,) or (n_targets, n_features) El número de trabajos para usar para el cálculo. Si -1 se usan todas las CPU. Esto solo proporcionará aceleración para n_targets> 1 y suficientes problemas grandes.
intercept_	Array Término independiente en el modelo lineal.

Tabla 15. *Atributos de la clase LinearRegression*

Métodos	Descripción
fit(X, y[, sample_weight])	Ajustar el modelo lineal.
get_params([deep])	Obtenga los parámetros para este estimador.
predict(X)	Predecir usando el modelo lineal.
score(X, y[, sample_weight])	Devuelve el coeficiente de determinación R ^ 2 de la predicción.
set_params(params)**	Establecer los parámetros de este estimador.

Tabla 16. *Métodos de la clase LinearRegression*

7.1 Ejemplo 1

Cargar librería.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Cargar datos.

```
boston = datasets.load_boston()

X = boston.data
y = boston.target
```

Dividir los datos en conjuntos de entrenamiento y prueba.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.3
```

Crear un objeto de regresión lineal.

```
regr = LinearRegression()
```

Entrene al modelo usando los conjuntos de entrenamiento.

```
regr.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normali
ze=False)
```

Hace las predicciones usando el conjunto de prueba.

```
regr.predict(X_test[:10])
```

```
array([24.87, 22.19, 24.75, 29.93, 30.77, 16.65, 11.6 , 28.92, 22.2
5, 18.13])
```

Ver los coeficientes del modelo de entrenamiento.

```
regr.coef_
```

```
array([ -0.13,   0.05,   0.06,   3.56, -15.8 ,   3.8 ,   0.01,  -1.
36,   0.35,  -0.01,  -0.89,   0.01,  -0.6 ])
```

Se observa que tuvo una predicción del 76%.

```
regr.score(X_test, y_test)
```

7.2 Ejemplo 2

Este ejemplo usa solo la primera característica del conjunto de datos de diabetes, para ilustrar un gráfico bidimensional de esta técnica de regresión. La línea recta se puede ver en la gráfica, que muestra cómo la regresión lineal intenta dibujar una línea recta que minimiza la suma residual de cuadrados entre las respuestas observadas en el conjunto de datos, y las respuestas predichas por la aproximación lineal.

Los coeficientes, la suma de cuadrados residuales y el puntaje de varianza también se calculan.

Cargar librerías.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

Cargar conjunto de datos de diabetes.

```
diabetes = datasets.load_diabetes()
```

Usa solamente una característica.

```
diabetes_X = diabetes.data[:, np.newaxis, 2]
```

Divide los datos en conjunto de entrenamiento/prueba.

```
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

Divida los objetivos en conjuntos de datos de entrenamiento/prueba.

```
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

Crea un objeto de regresión lineal.

```
regr = linear_model.LinearRegression()
```

Entrena al modelo usando los conjuntos de entrenamiento.

```
regr.fit(diabetes_X_train, diabetes_y_train)
```

Véase las predicciones para las primeras 10 instancias.

```
print linreg.predict(x[:10])
```

```
[30.01 25.03 30.57 28.61 27.94 25.26 23.   19.53 11.52 18.92]
```

Hace las predicciones usando el conjunto de prueba.

```
diabetes_y_pred = regr.predict(diabetes_X_test)
```

Los coeficientes.

```
print('Coefficients: \n', regr.coef_)
```

El error cuadrático medio.

```
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
```

Explicación de la puntuación de la varianza: 1 es la predicción perfecta.

```
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y
_pred))
```

Graficando resultado.

```
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=
3)

plt.xticks(())
plt.yticks(())

plt.show()
```

```
('Coefficients: \n', array([938.24]))
Mean squared error: 2548.07
Variance score: 0.47
```

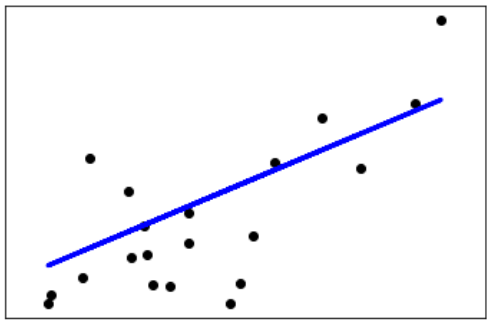

***Figura 4**. Imagen Regresión Lineal*

8 REGRESIÓN LOGÍSTICA

Clase LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1*)

Parámetros	Descripción
penalty	str, 'l1' o 'l2', valor predeterminado: 'l2' Se usa para especificar la norma utilizada en la penalización. Los solucionadores 'newton-cg', 'sag' y 'lbfgs' admiten solo l2 penalizaciones. Nuevo en la versión 0.19: penalización de l1 con solucionador de SAGA (permitiendo 'multinomial' + L1)
dual	bool, por defecto: False Formulación dual o primaria. La formulación dual solo se implementa para la penalización de l2 con el solucionador liblinear. Prefer dual = False cuando n_samples> n_features.
tol	float, por defecto: 1e-4 Tolerancia para detener los criterios
C	float, por defecto: 1.0 Inverso de la fuerza de regularización; debe ser un flotador positivo. Al igual que en las máquinas de vectores de soporte, los valores más pequeños especifican una regularización más fuerte
fit_intercept	bool, predeterminado: True Especifica si se debe agregar una constante (también conocida como sesgo o intersección) a la función de decisión
intercept_scaling	float, por defecto 1. Útil solo cuando se usa el solucionador 'liblinear' y self.fit_intercept está establecido en True. En este caso, x se convierte en [x, self.intercept_scaling], es decir, se agrega una característica "sintética" con un valor constante igual a intercept_scaling al vector de instancia. La intersección se convierte en intercept_scaling * synthetic_feature_weight
class_weight	dict o 'balanced', por defecto: None Pesos asociados con las clases en la forma {class_label: weight} . Si no se da, se supone que todas las clases tienen peso uno. El modo 'balanced'' usa los valores de y para ajustar automáticamente los pesos inversamente proporcionales a las frecuencias de clase en los datos de entrada como n_samples / (n_classes * np.bincount(y)) . Tenga en cuenta que estos pesos se multiplicarán con sample_weight (pasado a través del método de ajuste) si se especifica sample_weight.

Parámetros	Descripción
ramdon_state_	int, instancia de RandomState o None, opcional, por defecto: None La semilla del generador de números pseudoaleatorios para usar al mezclar los datos. Si int, random_state es la semilla utilizada por el generador de números aleatorios; Si la instancia de RandomState, random_state es el generador de números aleatorios; Si no es así, el generador de números aleatorios es la instancia de RandomState utilizada por np.random . Se usa cuando el solver == 'sag' o 'liblinear'.
solver:	str, {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, Predeterminado: Algoritmo 'liblinear' para usar en el problema de optimización. • Para conjuntos de datos pequeños, 'liblinear' es una buena opción, mientras que 'sag' y 'saga' son más rápidos para los grandes. • Para problemas multiclase, solo 'newton-cg', 'sag', 'saga' y 'lbfgs' manejan la pérdida multinomial; 'liblinear' está limitado a esquemas de uno contra el resto. • 'newton-cg', 'lbfgs' y 'sag' solo manejan la penalización de L2, mientras que 'liblinear' y 'saga' manejan la penalización de L1. Tenga en cuenta que la convergencia rápida 'sag' y 'saga' solo está garantizada en funciones con aproximadamente la misma escala.
max_iter	int, por defecto: 100 Útil solo para los solucionadores newton-cg, sag y lbfgs. Número máximo de iteraciones tomadas para que los solucionadores converjan.
multi_class	str, {'ovr', 'multinomial'}, predeterminado: 'ovr' La opción multiclase puede ser 'ovr' o 'multinomial'. Si la opción elegida es 'ovr', entonces un problema binario se ajusta para cada etiqueta. De lo contrario, la pérdida minimizada es la pérdida multinomial en toda la distribución de probabilidad. No funciona para el solucionador 'liblinear
verbose	int, default: 0 Para los solucionadores liblinear y lbfgs, establezca verbose a cualquier número positivo para verbosidad.
warm-start	bool, por defecto: False Cuando se establece en True, reutilice la solución de la llamada anterior para que se ajuste como inicialización, de lo contrario, simplemente borre la solución anterior.
n_jobs	int, por defecto: 1 Número de núcleos de CPU utilizados al paralelizar sobre clases si multi_class = 'ovr' ". Este parámetro se ignora cuando `` solver`` se establece en 'liblinear' independientemente de si se especifica 'multi_class' o no. Si se le da un valor de -1, se usan todos los núcleos.

Tabla 17. *Parámetros de la clase LogisticRegression*

Atributos	Descripción
coef_	Array de la forma (1, n_features) o (n_classes, n_features) Coeficiente de las funciones en la función de decisión. coef_ tiene forma (1, n_features) cuando el problema es binario. En particular, cuando multi_class = 'multinomial' , coef_ corresponde al resultado 1 (Verdadero) y -coef_ corresponde al resultado 0 (Falso).
intercept_	Array de la forma (1,) o (n_classes,) Interceptar (también conocido como sesgo) agregado a la función de decisión. Si fit_intercept está establecido en False, la intersección se establece en cero. intercept_ tiene forma (1) cuando el problema es binario. En particular, cuando multi_class = 'multinomial' , interceptar_ corresponde al resultado 1 (Verdadero) e -intercepto_ corresponde al resultado 0 (Falso).
n_iter	Array de la forma (n_classes,) o (1,) Número real de iteraciones para todas las clases. Si es binario o multinomial, devuelve solo 1 elemento. Para el solucionador liblinear, solo se proporciona el número máximo de iteraciones en todas las clases.

***Tabla 18**. Atributos de la clase LogisticRegression*

Métodos	Descripción
decision_function(X)	Predice los puntajes de confianza para las muestras.
densify()	Convertir matriz de coeficientes a formato de matriz densa.
fit(X, y[, sample_weight])	Ajuste el modelo de acuerdo con los datos de entrenamiento proporcionados.
get_params ([deep])	Obtenga los parámetros para este estimador.
predict (X)	Predecir etiquetas de clase para muestras en X.
predict_log_proba (X)	Registro de estimaciones de probabilidad.
predict_proba(X)	Estimaciones de probabilidad.
score(X, y[, sample_weight])	Devuelve la precisión medida en los datos y etiqueta de prueba de datos.
set_params (params)**	Establezca los parámetros de este estimador.
sparsify()	Convierte la matriz de coeficiente a formato disperso.

***Tabla 19**. Métodos de la clase LogisticRegression*

8.1 Ejemplo 1

Cargar librería.

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
```

Cargar datos.

```
iris = datasets.load_iris()
X = iris.data[:100,:]
y = iris.target[:100]
```

Estandarizar características.

```
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

Crea el objeto de Regresión logística.

```
clf = LogisticRegression(random_state=0)
```

Entrena el modelo.

```
clf.fit(X_std, y)
```

Predecir clase de observación.

```
clf.predict(new_observation)
```

```
array([1])
```

Ver probabilidades pronosticadas.

```
clf.predict_proba(new_observation)
```

```
array([[ 0.18823041,  0.81176959]])
```

8.2 Ejemplo 2

Cargar librerías.

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Cargar datos.

```
iris = datasets.load_iris()

X = iris.data
y = iris.target
```

Rehace la variable, manteniendo todos los datos en la categoría no es 2.

```
X = X[y != 2]
y = y[y != 2]
```

Ver las características.

```
X[0:5]
```

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

Ver los datos objetivos.

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1])
```

Separa los datos en conjuntos de prueba y entrenamiento, con el 30% de las muestras puestas en el conjunto de prueba.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.3, random_state=0)
```

Debido a que la penalización de regularización se compone de la suma del valor absoluto de los coeficientes, necesitamos escalar los datos para que los coeficientes estén todos basados en la misma escala.

Se crea un objeto escalar.

```
sc = StandardScaler()
```

Ajustar el escalador a los datos de entrenamiento y transformar.

```
X_train_std = sc.fit_transform(X_train)
```

Aplicar el escalador a los datos de prueba.

```
X_test_std = sc.transform(X_test)
```

La utilidad de L1 es que puede enviar coeficientes de característica a 0, creando un método para la selección de características. En el siguiente código ejecutamos una regresión logística con una penalización L1 cuatro veces, cada vez disminuyendo el valor de C. Deberíamos esperar que a medida que C disminuya, más coeficientes se conviertan en 0.

```
C = [10, 1, .1, .001]

for c in C:
    clf = LogisticRegression(penalty='l1', C=c)
    clf.fit(X_train, y_train)
    print('C:', c)
    print('Coefficient of each feature:', clf.coef_)
    print('Training accuracy:', clf.score(X_train, y_train))
    print('Test accuracy:', clf.score(X_test, y_test))
    print('')
```

```
C: 10
Coefficient of each feature: [[-0.0855264  -3.75409972  4.40427765  0.      ]]
Training accuracy: 1.0
Test accuracy: 1.0

C: 1
Coefficient of each feature: [[ 0.         -2.28800472  2.5766469   0.      ]]
Training accuracy: 1.0
Test accuracy: 1.0

C: 0.1
Coefficient of each feature: [[ 0.         -0.82310456  0.97171847  0.      ]]
Training accuracy: 1.0
Test accuracy: 1.0

C: 0.001
Coefficient of each feature: [[ 0.  0.  0.  0.]]
Training accuracy: 0.5
Test accuracy: 0.5
```

Observe que a medida que C disminuye, los coeficientes del modelo se vuelven más pequeños (por ejemplo, de 4.36276075 cuando C = 10 a .0.97175097 cuando C = 0.1),

hasta que en C = 0.001 todos los coeficientes son cero. Este es el efecto de la penalización de regularización cada vez más prominente.

9 K-MEANS

El algoritmo KMeans agrupa los datos tratando de separar las muestras en n grupos de igual varianza, minimizando un criterio conocido como inercia o suma de cuadrados dentro del grupo. Este algoritmo requiere que se especifique la cantidad de clústeres. Se adapta bien a una gran cantidad de muestras y se ha utilizado en una amplia gama de áreas de aplicación en muchos campos diferentes.

Clase KMeans

class sklearn.cluster.**KMeans**(*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto')*

Parámetros	Descripción
X	Conjunto de datos
y	Etiquetas para **X**
n_clusters	int, opcional, valor predeterminado = 8. Cantidad de centroides que se generarán.
init	{'k-means++', 'random' o un ndarray} Método para la inicialización, se predetermina a 'k-means++'
k-means++	Selecciona los centros de clúster iniciales para la agrupación k-means de una manera inteligente para acelerar la convergencia.
random	Elige k observaciones /filas) al azar a partir de los datos para los centroides iniciales.
ndarray	(n_clusters, n_features) De los centros iniciales.
max_iter	int, valor predeterminado = 300, Número máximo de iteraciones para una sola ejecución.
tol	float, valor predeterminado =1e-4 Tolerancia relativa con respecto a la inercia para declarar la convergencia.
precompute_distances	{'auto', True, False} Precálculo de distancias (más rápido, pero toma más memoria).
auto	No precálcula distancias si n_samples*n_clusters>12 millones.
True	Siempre precálcula las distancias
False	Nunca precálcula las distancias
verbose	int, valor predeterminado = 0

Parámetros	Descripción
random_state	int, instancia de RandomState o None, opcional, valor predeterminado: None
copy_x	boolean, valor predeterminado True Al calcular las distancias, es más preciso numéricamente centrar primero los datos. Si copy_x es True, entonces los datos originales no se modifican. Si es False, los datos originales se modifican y se vuelven a colocar antes de que la función retorne, pero se pueden introducir pequeñas diferencias numéricas al restar y luego agregar la medida de los datos.
n_jobs	int Número de trabajos a usar para el cálculo. Esto funciona al calcular cada una de las ejecuciones n_init en paralelo. If -1 se usan todas las CPUs. Si se da 1, no se utilizan ningún código de cómputo paralelo, lo cual es útil para la depuración. Para n_jobs por debajo de -1, se usan (n_cpus + 1 + n_jobs). Por lo tanto, para n_jobs = -2, se usan todas las CPU excepto una.
algorithm	"auto", "full" o "elkan", valor predeterminado = "auto"

Tabla 20. *Parámetros de la clase KMeans*

Atributos	Descripción
cluster_center_	Array, [n_clusters, n_features] Coordenadas de los centros de clúster .
labels_	Etiquetas de cada punto.
inertia_	float Suma de distancias cuadradas de las muestras a su centro de clúster más cercano.

Tabla 21. *Atributos de la clase KMeans*

Métodos	Descripción
fit(X, y)	Calcula el k-means clustering
fit_predict(X[,y])	Calcula los centros de clúster y pronostique el índice de clúster para cada muestra
fit_transform(X[,y])	Calcula los clúster y transforme X en un espacio de distancia de clúster.
get_params([deep])	Obtiene los parámetros para este estimador. **deep:** boolean, opcional (deep=True)
predict(X)	Predecir el clúster más cercano al que pertenece cada muestra.
score(X[,y])	Muestra que también aprendió el algorithm
set_params(** params)	Establece los parámetros del estimador.

Métodos	Descripción
transform(X)	Transforma X en un espacio de distancia de clúster.

***Tabla 22**. Métodos de la clase KMeans*

9.1 Ejemplo 1

Cargar librerías.

```
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn import metrics
```

Cargar datos.

```
iris = datasets.load_iris()

X = iris.data #Características
y = iris.target #Etiquetas
```

Se indica en cuántos grupos se va a clasificar y cuantas veces se va a mover el centroide.

```
km = KMeans(n_clusters=3, max_iter=3000)
```

Se entrena el algoritmo.

```
km.fit(X)
```

Se obtiene las predicciones de aquel grupo cree pertenecer las mediciones que hay en X.

```
predicciones = km.predict(X)
```

```
print(predicciones)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2,
       2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2,
       2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])
```

Se usan las etiquetas y las predicciones que se acaban de generar para ver que también aprendió.

```
score = metrics.adjusted_rand_score(y, predicciones)
```

```
print(score)
```

```
0.7302382722834697
```

Acertó el 73% de las veces.

9.2 Ejemplo 2

Cargar librerías

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

Cargar datos.

```
iris = datasets.load_iris()
X = iris.data
```

Se estandarizan las características.

```
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

Se crea un objeto k-mean.

```
clt = KMeans(n_clusters=3, random_state=0, n_jobs=-1)
```

Se entrena el modelo.

```
clt.fit(X_std)
```

Mostrar Pertenencia al clúster de cada observación.

```
clt.labels_
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
```

```
       1, 1, 1, 1, 0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0,
2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2,
2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
0, 2, 2,
       0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2,
0, 0, 0,
       2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2], dtype=int32)
```

Se crean nuevas observaciones.

```
new_observation = [[0.8, 0.8, 0.8, 0.8]]
```

Se predice el grupo de observación.

```
clt.predict(new_observation)
```

```
array([0], dtype=int32)
```

Ver centro de cada clúster.

```
model.cluster_centers_
```

```
array([[ 1.13597027,  0.09659843,  0.996271  ,  1.01717187],
       [-1.01457897,  0.84230679, -1.30487835, -1.25512862],
       [-0.05021989, -0.88029181,  0.34753171,  0.28206327]])
```

10 VECINOS MÁS CERCANOS

Clase KNeighborsClassifier

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights
='uniform', algorithm='auto',leaf_size=30, p=2, metric='minkowski',
 metric_params=None, n_jobs=1, **kwargs)
```

Clase KNeighborsRegressor

```
class sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, weights=
'uniform', algorithm='auto',leaf_size=30, p=2, metric='minkowski',
metric_params=None, n_jobs=1, **kwargs)
```

Parámetros	Descripción
X	Conjunto de datos
y	Etiquetas para **X**
n_neighbors	int, número de vecinos
weights	str o callable, opcional (valor predeterminado = 'uniform') función de peso utilizada en la predicción.
uniform	pesos uniformes. Todos los puntos en cada barrio se ponderan por igual.
distance	Puntos de peso por el inverso de su distancia. en este caso, los vecinos más cercanos de un punto de consulta tendrán una mayor influencia que los vecinos que están más lejos.
[callable]	una función definida por el usuario que acepta una matriz de distancias y devuelve una matriz de la misma forma que contiene los pesos.
algorithm	{'auto', 'ball_tree', 'Kd_tree', 'brute'}, opcional
brute	La implementación de búsqueda de vecinos más ingenua implica el cálculo de la fuerza bruta de las distancias entre todos los pares de puntos en el conjunto de datos: para las *N* muestras (n_samples) en *D* dimensiones (n_features), este enfoque se escala como $O[DN^2]$. Para muestras pequeñas. Si *N* crece se vuelve rápidamente inviable.
kd_tree	Aborda las ineficiencias computacionales del enfoque de la fuerza bruta. la búsqueda de un vecino más cercano se puede reducir a $O[DNlog(N)]$ o mejorar. Esto es una mejora significativa sobre la fuerza bruta para grandes *N*. Es muy rápido para bajas dimensiones ($D < 20$), pero se vuelve ineficiente a medida que *D* crece.
ball_tree	Aborda las ineficiencias de los arboles KD en dimensiones superiores.
leaf_size	int, opcional (por defect = 30)
p	integer, opcional (valor predeterminado = 2) Parámetro de potencia para la métrica de minkowski. Cuando p=1, esto es equivalente a utilizar manhattan_distance (l1), y euclidean_distance (l2) para p = 2. Para p arbitraria, se usa minkowski_distance (l_p).
metric	str o callable, valor predeterminado 'minkowski' la métrica de distancia a usar para el árbol. La métrica predeterminada es minkowski, y con p = 2 es equivalente a la métrica euclidiana estándar.
metric_params	dic, opcional (valor predeterminado = None) Argumentos claves adicionales para la función métrica.
n_jobs	int, opcional (predeterminado = 1) La cantidad de trabajos paralelos que se ejecutan para la búsqueda de vecinos, Si -1, entonces la cantidad de trabajos se establece en la cantidad de núcleos de CPU. No afecta el método **fit**.
mode	{'connectivity', 'distance'}, opcional

Parámetros	Descripción
	Tipo de matriz devuelta: 'connectivity' devolverá la matriz de conectividad con unos y ceros, en 'distance' los bordes son distancia euclídea entre puntos.
return_distance	boolean, opcional. El valor predeterminado es True. Si es falso, la distancia no será devuelta.
sample_weight	Peso de muestra, opcional.

***Tabla 23**. Parámetros de la clase KNeighborsClassifier- KneighborsRegressor*

Métodos	Descripción
fit(X, y)	Ajustar el modelo usando X como datos de entrenamiento e y como valores objetivo. Sirve para entrenar.
get_params([deep])	Obtenga los parámetros para este estimador. **deep:** boolean, opcional (deep=True)
kneighbors([X, n_neighbors, return_distance])	Encuentra los vecinos de un punto.
kneighbors_graph([X, n_neighbors, mode])	Calcula el gráfico (ponderado) de k-vecinos para los puntos en X. Devuelve una matriz **A** dispersa en formato CRS, de la forma [n_sample, n_samples_fit], s_samples_fit es el número de muestras en los datos ajustados. **A**[i, j] se le asigna el peso del borde que conecta i a j.
predict(X)	Predecir las etiquetas de clase para los datos proporcionados
predict_proba(X)	Estimaciones de probabilidad de retorno para los datos de prueba X. Devuelve **p**, en forma [n_samples, n_classess] o una lista de n_outputs de tales arreglos si n_outputs>1. Las probabilidades de clase de las muestras de entradas. Las clases se ordenan por orden lexicográfico.
score(X, y [, sample_weight])	Muestra que también aprendió el algorithm.
set_params(** params)	Establece los parámetros del estimador.

***Tabla 24**. Métodos de la clase KNeighborsClassifier- KNeighborsRegressor*

10.1 Clasificación de vecinos más cercano

Implementa el aprendizaje basado en los k vecinos más cercanos de cada punto de consulta, donde k se especifica un valor entero especificado por el usuario

10.1.1 Ejemplo 1

Cargar librería

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
```

Cargar datos.

```
iris = datasets.load_iris()
```

Se separa los datos en un conjunto de entrenamiento y de prueba.

```
X_train, X_test, y_train,
y_test=train_test_split(iris.data,iris.target,test_size=0.33)
```

X para entrenar, es una matriz de 100x4, 100 flores con sus 4 mediciones.

```
print(X_train.shape)
```

```
(100, 4)
```

Y para entrenar, es un vector de 100 etiquetas correspondientes a cada elemento de x, es decir, cada una de las clasificaciones.

```
print(y_train.shape)
(100,)
```

Se considera 5 vecinos para clasificar.

```
knn=KNeighborsClassifier(n_neighbors=5)
```

Se entrena el algoritmo.

```
knn.fit(X_train, y_train)
```

Se puede ver que el algoritmo aprendió bastante bien sobre un 98% del conjunto de prueba.

```
knn.score(X_test, y_test)
```

```
0.92
```

Se tienen 4 mediciones y se quiere que nos diga a qué clasificación pertenece.

```
print(knn.predict([[1.2,3.4,5.6,7.1]]))
```

```
array([2])
```

Se puede apreciar que pertenece al elemento 2 del arreglo; ‘virginica’.

```
print(iris.target_names)
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='|S10')
```

10.1.2 Ejemplo 2

Uso de muestra de la clasificación de vecinos más cercanos. Trazará los límites de decisión para cada clase.

Cargar librerías

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

Cargar datos.

```
iris = datasets.load_iris()
```

En x solo se toma coordenadas 2D.

```
X = iris.data[:, :2]
y = iris.target
```

Tamaño del paso de la malla.

```
h = .02
```

Crea mapa de color.

```
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

Se crea una instancia de vecinos clasificador y se ajusta los datos.

```
for weights in ['uniform', 'distance']:
    clf = neighbors.KNeighborsClassifier(n_neighbors = 15,
                    weights=weights)

    clf.fit(X, y)
```

Traza el límite de decisión. Para eso se asignara un color a cada punto en la malla *[x_min, x_max] * [y_min, y_max]*

```
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

```
    np.arange(y_min, y_max, h))

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

Pone el resultado en una trama de color.

```
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

Traza también los puntos de entrenamientos.

```
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))
plt.show()
```

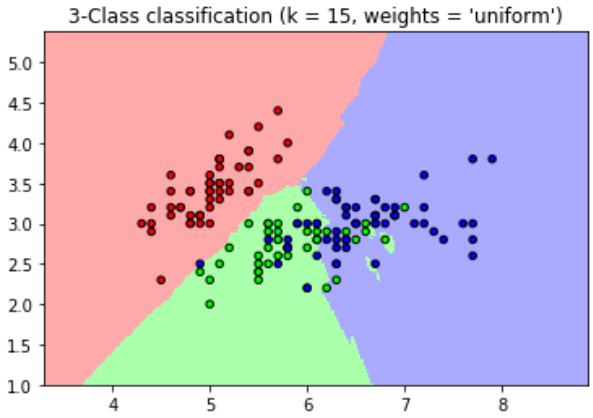

***Figura 5**. Clasificación de vecinos más cercano con peso uniforme*

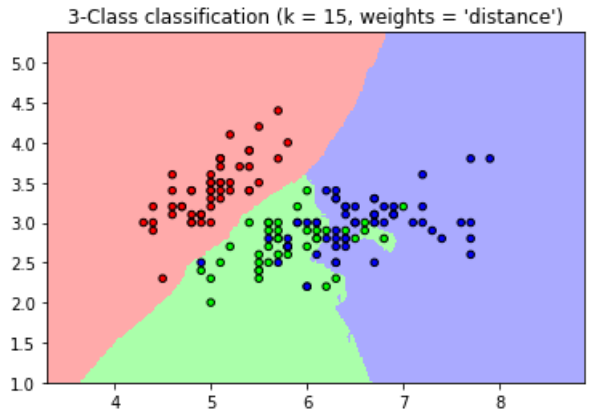

***Figura 6.** Clasificación de vecinos más cercano con peso distancia*

10.2 Regresión de vecinos más cercano

La regresión basada en vecinos se puede usar en casos donde las etiquetas de datos son variables continuas en lugar de discretas. La etiqueta asignada a un punto de consulta se calcula según la media de las etiquetas de sus vecinos más cercanos.

10.2.1 Ejemplo 1

Cargar librerías

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Cargar datos.

```
boston=load_boston()
```

Se separa los datos en un conjunto de entrenamiento y de prueba.

```
X_train, X_test, y_train, y_test = train_test_split(boston.data, bo
ston.target)
```

X para entrenar, es una matriz de 379x13, 379 casa con sus 13 características.

```
print(X_train.shape)
```

```
(379, 13)
```

Para hacer pruebas se usa un vector de 127 etiquetas correspondientes a cada elemento de x.

```
print(y_train.shape)
```

```
(127,)
```

Se considera 5 vecinos.

```
knn=KNeighborsRegressor(n_neighbors=5)
```

Se entrena el algoritmo.

```
knn.fit(X_train, y_train)
```

Se puede ver que el algoritmo aprendió muy mal, solo pudo hacer una predicción del 46%.

```
print(knn.score(X_test, y_test))
```

```
0.46086697006170485
```

Si se usa regresión lineal, sobre los mismos datos, se puede apreciar que aprende mucho mejor, con un 70% de respuestas correctas.

```
rl=LinearRegression()
rl.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normali
ze=False)
```

```
rl.score(X_test, y_test)
```

```
0.7078847630415585
```

10.2.2 Ejemplo 2

Este ejemplo muestra el uso del estimador de salida múltiple para completar imágenes. El objetivo es predecir la mitad inferior de una cara dada su mitad superior.

Cargar librerías.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces
from sklearn.utils.validation import check_random_state
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV
```

Cargar datos.

```
data = fetch_olivetti_faces()
targets = data.target

data = data.images.reshape((len(data.images), -1))
train = data[targets < 30]
test = data[targets >= 30] #Prueba sobre personas independientes
```

Prueba en un subconjunto de personas.

```
n_faces = 5
```

```
rng = check_random_state(4)
face_ids = rng.randint(test.shape[0], size=(n_faces, ))
test = test[face_ids, :]

n_pixels = data.shape[1]
```

Mitad superior de las caras.

```
X_train = train[:, :(n_pixels + 1) // 2]
```

Mitad inferior de las caras.

```
y_train = train[:, n_pixels // 2:]
X_test = test[:, :(n_pixels + 1) // 2]
y_test = test[:, n_pixels // 2:]
```

Estimadores de ajuste.

```
ESTIMATORS = {
    "Extra trees": ExtraTreesRegressor(n_estimators=10, max_features=32,
                                       random_state=0),
    "K-nn": KNeighborsRegressor(),
    "Linear regression": LinearRegression(),
    "Ridge": RidgeCV(),
}

y_test_predict = dict()
for name, estimator in ESTIMATORS.items():
    estimator.fit(X_train, y_train)
    y_test_predict[name] = estimator.predict(X_test)
```

Traza las caras completas.

```
image_shape = (64, 64)

n_cols = 1 + len(ESTIMATORS)
plt.figure(figsize=(2. * n_cols, 2.26 * n_faces))
plt.suptitle("Face completion with multi-output estimators", size=16)

for i in range(n_faces):
    true_face = np.hstack((X_test[i], y_test[i]))
    if i:
        sub = plt.subplot(n_faces, n_cols, i * n_cols + 1)
    else:
        sub = plt.subplot(n_faces, n_cols, i * n_cols + 1,
                          title="true faces")

    sub.axis("off")
    sub.imshow(true_face.reshape(image_shape),
               cmap=plt.cm.gray,
               interpolation="nearest")
```

```
    for j, est in enumerate(sorted(ESTIMATORS)):
        completed_face = np.hstack((X_test[i], y_test_predict[est][i]))
        if i:
            sub = plt.subplot(n_faces, n_cols, i * n_cols + 2 + j)
        else:
            sub = plt.subplot(n_faces, n_cols, i * n_cols + 2 + j,
                              title=est)
        sub.axis("off")
        sub.imshow(completed_face.reshape(image_shape),
                   cmap=plt.cm.gray,
                   interpolation="nearest")
plt.show()
```

***Figura 7**. Finalización de cara con un estimador de salida múltiple*

REFERENCIAS

http://isc.utp.edu.co/archivos/materia-62-is842-computacion-blanda.pdf
https://github.com/pagutierrez/tutorial-scikit-learn
https://drive.google.com/drive/folders/0B6XZi94jMWhuNkNKSWw5SVlYbDQ
https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico
https://relopezbriega.github.io/blog/2015/10/10/machine-learning-con-python/
https://medium.com/@ricardo.guerrero/frameworks-de-deep-learning-un-repaso-antes-de-acabar-el-2016-5b9bf5b9f9af
file:///G:/descargas/programming.collective.intelligence.aug.2007%20.pdf
http://choonsiong.com/public/books/Big%20Data/Data%20Science%20from%20Scratch.pdfhttps://medium.com/machine-learning-for-humans/how-to-learn-machine-learning-24d53bb64aa1
https://noemagico.blogia.com/2006/091301-la-investigaci-n-descriptiva.php
https://arxiv.org/abs/1309.0238
http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html
https://www.enriquedans.com/2016/10/inteligencia-artificial-y-machine-learning-como-nueva-frontera.html
http://www.bautistasanz.com/machine-learning-aplicable-los-sistemas-control/
https://www.expoelearning.com/machine-learning-inteligencia-artificial/
http://scikit-learn.org/stable/datasets/#toy-datasets
http://archive.ics.uci.edu/ml/index.php
Scikit-learn: Aprendizaje automático en Python , Pedregosa et al. , JMLR 12, pp. 2825-2830, 2011.
https://github.com/amueller/scipy-2017-sklearn
https://github.com/krasserm/machine-learning-notebooks/blob/master/ml-ex4.ipynb
https://github.com/chrisalbon/notes/tree/master/docs/machine_learning
https://github.com/scikit-learn/scikit-learn/tree/a24c8b464d094d2c468a16ea9f8bf8d42d949f84/sklearn
https://github.com/amueller/scipy-2017-sklearn/blob/master/notebooks/03.Data_Representation_for_Machine_Learning.ipynb